

Docket No.: K-254

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

#2  
JC796 U.S. PRO  
09/768301  
01/25/01  


In re Application of :

Sang Kyun CHA,  
Ju Chang LEE and Ki Hong KIM :

Serial No.: New U.S. Patent Application :

Filed: January 25, 2001 :

For: A METHOD AND SYSTEM FOR HIGHLY-PARALLEL LOGGING  
AND RECOVERY OPERATION IN MAIN-MEMORY  
TRANSACTION PROCESSING SYSTEMS

TRANSMITTAL OF CERTIFIED PRIORITY DOCUMENT

Assistant Commissioner of Patents  
Washington, D. C. 20231

Sir:

At the time the above application was filed, priority was claimed based on the following application:

Korean Patent Application No. 2000/31166, filed June 7, 2000.

A copy of each priority application listed above is enclosed.

Respectfully submitted,  
FLESHNER & KIM, LLP

Daniel Y.J. Kim  
Registration No. 36,186

P. O. Box 221200  
Chantilly, Virginia 20153-1200  
703 502-9440  
Date: January 25, 2001  
DYK/ieh

JC796 U.S. PTO  
09/16/01  
01/25/01

대한민국 특허청  
KOREAN INDUSTRIAL  
PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto  
is a true copy from the records of the Korean Industrial  
Property Office.

출원번호 : 특허출원 2000년 제 31166 호  
Application Number

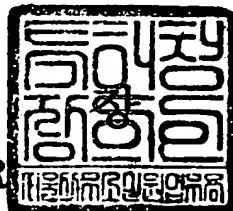
출원년월일 : 2000년 06월 07일  
Date of Application

출원인 : 차상균 외 2명  
Applicant(s)

2000 년 12 월 15 일

특허청

COMMISSIONER



1020000031166

2000/12/1

【서류명】	특허출원서
【권리구분】	특허
【수신처】	특허청장
【제출일자】	2000.06.07
【발명의 명칭】	트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜로깅 방법
【발명의 영문명칭】	A Logging Method for Commutative and Associative Recovery Operation in Transaction Processing Systems
【출원인】	
【성명】	차상균
【출원인코드】	4-1998-035573-1
【출원인】	
【성명】	김기홍
【출원인코드】	4-1998-055795-1
【출원인】	
【성명】	이주창
【출원인코드】	4-2000-027103-3
【대리인】	
【성명】	김영철
【대리인코드】	9-1998-000040-3
【포괄위임등록번호】	2000-031923-5
【포괄위임등록번호】	2000-031927-4
【포괄위임등록번호】	2000-031584-4
【대리인】	
【성명】	김순영
【대리인코드】	9-1998-000131-1
【포괄위임등록번호】	2000-031925-0
【포괄위임등록번호】	2000-031926-7
【포괄위임등록번호】	2000-031583-7
【발명자】	
【성명】	차상균
【출원인코드】	4-1998-035573-1

1020000031166

2000/12/1

【발명자】

【성명】 김기홍

【출원인코드】 4-1998-055795-1

【발명자】

【성명】 이주창

【출원인코드】 4-2000-027103-3

【심사청구】

【취지】

청구

특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정  
에 의한 출원심사를 청구합니다. 대리인  
김영철 (인) 대리인  
김순영 (인)

【수수료】

【기본출원료】 20 면 29,000 원

【가산출원료】 31 면 31,000 원

【우선권주장료】 0 건 0 원

【심사청구료】 20 항 749,000 원

【합계】 809,000 원

【감면사유】 개인 (70%감면)

【감면후 수수료】 242,700 원

### 【요약서】

#### 【요약】

본 발명은 트랜잭션 단위의 원자성 또는 지속성을 보장해야 하는 트랜잭션 프로세싱 시스템을 위한 로깅 방법에 있어서, 로그의 본체는 로그 레코드의 변경 전 이미지와 변경 후 이미지간의 XOR 결과로 이루어지며 리두나 언두 연산은 로그 본체와 그 로그코드에 코드가 생성되었던 위치에 저장되어 있는 원본 데이터 간의 XOR 연산을 통해 이루어지는 것을 특징으로 하는 디퍼런셜 로깅 방법에 관한 것이다. 본 발명에서 제안하는 디퍼런셜 로깅 방식의 다음과 같은 특성들을 이용할 경우, 전자 상거래, 통신 시스템, 풍차장 제어 등 주 메모리에서 데이터를 관리하는 응용 시스템들의 로깅 및 재시작 성능을 크게 향상시킬 수 있다. 먼저, XOR 연산은 교환성과 결합성을 가지고 있으므로 리두 또는 언두 연산 시 로그 레코드의 생성 순서와 관계없이 임의의 순서로 연산을 하더라도 정확한 복구가 가능하다. 따라서, 이러한 디퍼런셜 로깅의 교환성과 결합성을 이용할 경우 효과적인 병렬 로깅 및 병렬 재시작 기법이 가능하다. 또한, 디퍼런셜 로깅에서는 리두와 언두 연산 간의 구분이 필요 없기 때문에 단일 경로 재시작 기법을 가능하게 하는 근거를 마련해 준다. 뿐만 아니라, 디퍼런셜 로깅은 변경 전 이미지와 변경 후 이미지를 저장하는 물리적 로깅과 달리 이들간의 차이만을 저장하므로 로그량을 물리적 로깅에 비해 로그량을 절반 가까이 줄일 수 있다.

#### 【대표도】

도 1

1020000031166

2000/12/1

한국은 디퍼런스를 두고 있다.

한국정치

## 【명세서】

### 【발명의 명칭】

트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법

{A Logging Method for Commutative and Associative Recovery Operation in Transaction Processing Systems}

### 【도면의 간단한 설명】

도1은 디퍼런셜 로그의 구조 및 생성과정에 관한 도면,

도2는 디퍼런셜 로그에 대한 리두와 언두에 관한 도면,

도3은 물리적 로깅을 이용한 회복과정과 디퍼런셜 로깅을 이용한 회복과정의 차이에 대한 도면,

도4는 주 메모리에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템의 논리적 구조에 관한 도면,

도5는 주 메모리에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서 퍼지 체크포인팅을 디퍼런셜 로깅과 함께 사용하기 위해서 필요한 갱신 연산 알고리즘,

도6은 주 메모리에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서 디퍼런셜 로깅을 사용하기 위해서 필요한 퍼지 체크포인팅 알고리즘,

도7은 주 메모리에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서 퍼지 체크포인팅을 디퍼런셜 로깅과 함께 사용하기 위해서 필요한 재시작 알고리즘,

도8은 디스크에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템의 논리적 구조에 관한 도면,

도9는 디스크에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서 디퍼런셜

로깅을 사용하기 위해 필요한 갱신 연산 알고리즘,

도10은 디스크에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서 디퍼런셜로깅을 사용하기 위해 필요한 재시작 알고리즘,

도11은 주 메모리에 원본 데이터를 보관하고 퍼지 체크포인팅을 사용하는 트랜잭션 프로세싱 시스템에서의 단일 경로 재시작 알고리즘,

도12는 주 메모리에 원본 데이터를 보관하고 일관성 체크포인팅을 사용하는 트랜잭션 프로세싱 시스템에서의 단일 경로 재시작 알고리즘,

도13은 디스크에 원본 데이터를 보관하는 트랜잭션 프로세싱 시스템에서의 단일 경로 재시작 알고리즘,

도14는 트랜잭션 별로 로그를 분배하는 병렬적 로깅 구조에 대한 도면,

도15는 트랜잭션 별로 로그를 분배하는 병렬적 로깅 구조에서 병렬적으로 재시작하는 구조에 관한 도면,

도16은 하나의 로그 디스크가 있을 때, 로그 분배 쓰레드를 이용한 병렬적 재시작 구조에 관한 도면,

도17은 하나의 로그 디스크가 있을 때, 로그 분배 쓰레드가 없는 병렬적 재시작 구조에 관한 도면,

도18은 복수의 로그 디스크와 복수의 백업 디스크를 이용한 병렬적 재시작 구조에 관한 도면,

도19는 백업 데이터를 원본 데이터에 적용하는 과정과 로그를 원본 데이터에 적용하는 과정을 병렬적으로 수행하는 알고리즘.

#### 【발명의 상세한 설명】

##### 【발명의 목적】

##### 【발명이 속하는 기술분야 및 그 분야의 종래기술】

<20> 본 발명은 트랜잭션 단위의 원자성과 지속성을 보장해야 하는 트랜잭션 프로세싱 시스템에서의 로깅 방식에 관한 것으로 종래의 로깅 방식들이 교환성(commutativity)과 결합성(associativity)을 갖추지 못한 것과는 달리 디퍼런셜로깅(differential logging)이라는 새로운 로깅 방법을 제안하여 로그 레코드의 생성 순서와 관계없이 임의의 순서로 리두(redo)나 언두(undo)를 하더라도 정확한 데이터베이스로의 회복이 가능한 것을 특징으로 하는 로깅 방법에 관한 것이다.

<21> 트랜잭션 프로세싱 시스템은 오류가 발생하더라도 원본 데이터의 일관성 및 지속성을 유지해야 한다. 이를 위해 대부분의 트랜잭션 프로세싱 시스템들은 시스템이 동작하는 중에 로그라 부르는 시스템의 회복에 필요한 여러 가지 정보를 안전한 저장장치에 기록하며, 오류가 발생한 경우에는 로그를 이용하여 원본 데이터의 일관성과 지속성을 유지시킨다. 이때 기존의 로깅 방식을 따를 경우 동일한 원본 데이터 영역에서 발생한 로그들은 회복 과정에서 그 생성 순서대로, 즉 먼저 생성된 로그가 먼저 리두 되어야지 일관된 시스템 상태로의 복구가 가능하다. 이러한 기존 로깅 방식의 비결합성 및 비 교환

성은 회복 시스템 구성에서 있어서 많은 제약을 받게 된다.

<22> 예를 들어, 대부분의 연산을 주 메모리 상에서 처리하는 주 메모리 데이터베이스 관리 시스템의 경우 회복을 위한 로그 디스크 접근은 시스템 성능 상 큰 병목으로 작용 한다. 따라서, 이러한 병목을 상쇄하기 위해서는 복수의 로그 디스크를 사용할 필요가 있는데, 기존의 로깅 방식을 사용할 경우 정확한 회복을 위해 써는 로깅 성능이나 재시작 성능 중 하나의 성능 저하가 불가피하다. 만약 로깅 시 각 로그들을 임의로 각 로그 디스크에 분배하게 되면, 재시작 시 각 로그 디스크에 흩어져 있는 로그들 간의 생성·우선 순위를 파악하기 위한 오버헤드가 필요하기 때문에 재시작 성능의 저하가 불가피하다. 또한, 그러한 생성 우선 순위 파악을 위한 오버헤드를 없애기 위해 데이터베이스를 몇 개의 영역으로 나누고서 한 영역에서 발생한 로그는 그 영역에 해당하는 로그 디스크에만 저장하도록 할 경우 역시, 로깅 성능의 저하가 불가피하다. 왜냐하면, 데이터베이스의 일부분만을 접근하는 트랜잭션들이 많을 경우 복수의 로그 디스크 중 일부의 로그 디스크만을 사용하게 되는 결과를 낳게 되기 때문이다. 또한, 한 트랜잭션이 여러 데이터베이스 영역을 접근한 경우 그 트랜잭션을 완료하기 위해서는 여러 로그 디스크에 대한 접근이 필요하기 때문이다. 물론, RAID를 로그 디스크로 쓰는 방법이 있는데, 이 방법을 고성능 다중 프로세서 환경에서 사용할 경우 로그 버퍼에서 발생하는 잠금 충돌이 심각한 오버헤드로 작용할 수 있다. 그리고, RAID는 복수의 물리적 디스크를 하나의 논리적 디스크와 같이 동작하도록 하는 장치이므로 위의 논의와는 별개로 취급할 수 있다.

#### 【발명이 이루고자 하는 기술적 과제】

<23> 본 발명은 기존의 병렬적 회복 기법들이 가지는 한계인 리두나 언두 연산에 있어서

의 교환성과 결합성의 부재를 극복하기 위하여 디퍼런셜 로깅이라는 새로운 로깅 방법을 제안하고자 한다. 이 회복 기법 하에서는 리두나 언두 시 XOR연산을 연산을 수행하는데 임의의 이진수에 대해 XOR 연산은 교환성과 결합성을 가지고 있으므로 로그 레코드의 생성 순서와 관계없이 리두나 언두를 하더라도 정확한 시스템 상태의 복구가 가능하다.

<24> 또한 본 발명에서는 위와 같은 디퍼런셜 로깅의 성질을 이용하여 보다 효과적인 병렬적 재시작 기법을 제안하며, 한번의 로그 스캔만으로 리두와 언두 연산을 동시에 행할 수 있는 단일 경로 재시작 기법을 제안하고자 한다.

#### 【발명의 구성 및 작용】

<25> 상기한 목적을 달성하기 위하여 본 발명에 의한 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법은 로그 레코드의 본체는 변경된 데이터의 변경 전 이미지와 변경 후 이미지간의 XOR 결과로 이루어지며 리두 연산 또는 언두 연산은 상기 로그 레코드의 본체와 상기 로그 레코드가 생성되었던 위치에 저장되어 있는 데이터간의 XOR 연산을 통해 이루어지는 것을 특징으로 한다.

<26> 이하에서 첨부된 도면을 참조하여 본 발명에 의한 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복연산을 위한 로깅방법을 상세하게 설명한다.

<27> 도1은 디퍼런셜 로깅의 구조 및 생성 과정에 대한 것이다. 로그 레코드는 로그 헤더와 로그 본체로 구성된다. 로그 헤더는 로그의 생성 순서를 나타내며 주로 로그 딕스코의 물리적 주소값이 저장되는 'LSN(Log Sequence Number)'(11), 본 로그 레코드를 만들어낸 트랜잭션에 의해 가장 최근에 만들어진 로그 레코드의 아이디인 'PrevLSN'(12), 본 로그 레코드를 생성한 트랜잭션의 아이디 정보를 나타내는 'TID'(13), 로그 레코드의 타

입을 나타내는 정보인 'type(14)', 퍼지 체크포인트와 함께 사용 시 로그와 변경된 페이지 사이의 관계를 나타내는 'UCC(update counter during checkpointing)''(15), 변경된 페이지의 위치를 나타내는 'PageID'(16), 변경된 슬롯의 위치를 나타내는 'Slot ID'(17), 변경된 슬롯의 길이를 나타내는 'Size'(18)로 구성된다. 이러한 구조는 하나의 데이터베이스는 여러 개의 동일한 크기를 가진 페이지로, 하나의 페이지는 여러 개의 동일한 크기를 가진 슬롯으로 구성되어 있을 때를 가정한 것이다. 다른 형태의 저장 구조를 지닌 경우에는 약간의 변경을 통해 손쉽게 적용할 수 있다. 이때 슬롯은 갱신 연산 및 로그 생성의 최소 단위가 된다. 로그의 본체(19)는, 기존의 물리적 로깅 방식이 갱신연산에 의해 변경되기 전의 이미지와 변경 후의 이미지를 모두 저장하는데 반해, 디퍼런셜 로깅에서는 변경 전 이미지와 변경 후 이미지간의 XOR 결과만을 로그 본체에 저장한다. 예를 들어 변경전의 이미지가 0011이고 변경 후의 이미지가 0101이라면, 기존의 로깅 방식은 이 두 개의 이미지를 모두 저장하는데 반해 디퍼런셜 로깅에서는 두 개의 이미지간의 XOR 결과인 0110만을 로그 본체에 저장하게 되는 것이다.

<28>         도2는 디퍼런셜 로깅을 이용한 리두나 언두 연산을 나타낸 그림이다. 로그 헤더에는 'PageID'와 'Slot ID'로부터 해당 로그가 발생한 위치를 파악한 다음, 거기에 존재하는 원본 데이터와 로그 본체에 저장된 데이터간의 XOR 연산을 통해 리두나 언두가 수행되게 된다.

<29>         도 3은 디퍼런셜 로깅과 물리적 로깅의 차이를 나타낸 도면이다. 물리적 로깅 방식이 갱신연산에 의해 변경되기 전의 이미지와 변경 후의 이미지를 모두 저장하는 데 반해, 디퍼런셜 로깅에서는 변경 전 이미지와 변경 후 이미지간의 XOR 결과를 로그 본체에 저장하게 된다. 또한, 물리적 로깅은 항상 그 생성 순서대로 리두 되어야지만 정확한

결과 값을 얻을 수 있다. 이는 물리적 로깅 방식이 교환성과 결합성을 갖추고 있지 못하기 때문이다. 반면에, 디퍼런셜 로깅은 그 생성 순서와 상관없이 리두 하더라도 정확한 결과값을 얻을 수 있는 것을 도3으로부터 알 수 있다.

<30> 이러한 디퍼런셜 로깅은 주 메모리 상에서 원본 데이터를 관리하는 시스템이나 디스크 상에서 원본 데이터를 관리하는 시스템에 모두 적용 가능하다. 도4는 주 메모리 상에서 원본 데이터를 관리하는 트랜잭션 프로세싱 시스템의 구조이다. 주 메모리 상에서 원본 데이터를 관리하는 시스템에서, 만약 그 시스템이 일관성 체크포인팅을 사용하고 있다면 체크 포인팅 도중에 로그들이 생성되지 않으므로 상기한 디퍼런셜 로깅을 곧바로 적용하는 것이 가능하다.

<31> 만약 시스템이 퍼지 체크포인팅을 사용하고 있다면 곧바로 상기한 바와 같은 디퍼런셜 로깅을 곧바로 적용할 수 없다. 퍼지 체크포인팅은 상기한 일관성 체크포인팅과는 달리 체크포인팅 도중에도 다른 트랜잭션이나 연산들의 수행을 허용하기 때문이다. 따라서 퍼지 체크포인팅을 사용하고 있다면 체크포인팅 도중에도 로그들이 생성될 수 있다. 이와 같은 경우에 시스템 재시작 시 어떤 갱신 연산에 의한 결과가 반영되어 백업된 폐이지에 대해 그 갱신 연산에 의해 생성된 로그를 또 다시 적용할 수 있는 경우가 발생하게 된다. 즉 사전 이미지와 사후이미지 간의 디퍼런셜 로깅이 적용된 이후 사후이미지에다가 다시 디퍼런셜 로깅을 적용해버리는 결과가 되는 경우이다. 앞서 설명한 바와 같이 디퍼런셜 로깅은 사전 이미지에 XOR 연산을 하여야지 사후 이미지에 XOR 연산을 부정확한 결과가 나오게 된다.

<32> 본 발명에서는 퍼지 체크포인팅을 사용할 경우에 이러한 문제를 피하기 위해, 체크포인팅 도중에 갱신 연산이 발생하면, 그 갱신 연산에 의해 생성된 로그 레코드에 일종

의 일련번호를 부여한 후, 그 일련 번호를 데이터베이스 페이지에도 기록해 두어, 시스템 재시작 시 리두를 할 필요가 있는지 없는지 판단할 수 있도록 한다. 이와 같은 일련 번호에 해당하는 것이 UCC이다. 페이지의 UCC는 체크포인팅 이루어지는 동안 생신 연산 ~~을~~ 이 이루어지는 경우에 하나씩 증가되며, 체크포인트 매니저에 의해 체크포인트 마지막 로그 레코드를 기록할 때 리셋된다. 로그 레코드를 페이지에 적용할 것인가의 여부는 로그 레코드의 UCC와 로그를 적용하려는 페이지의 UCC를 비교하여 결정한다. 로그 레코드 ~~을~~의 UCC가 그 로그 레코드를 적용하려 하는 페이지의 UCC와 같거나 더 작은 번호를 갖고 있다면, 그 로그가 그 페이지에 적용되어서는 안된다.

<33> 퍼지 체크포인팅을 사용할 경우에 또 하나의 문제점은 생신 연산이 이루어지고 난 후, 그 생신연산에 대한 로그 일련번호를 기록해주는 사이에 그 해당 데이터가 체크 포인팅에 의해 디스크로 내려가 버릴 수 있다. 이런 경우, 페이지 일련 번호와 페이지 내용 사이의 일관성, 즉 사전 이미지와 사후 이미지가 혼합이 되어 카피가 되어버릴 수 있고 시스템 붕괴가 일어나는 경우 등에는 페이지 복구가 이루어질 수 없다.

<34> 본 발명에서는 이와 같은 문제가 일어나지 않도록 체크포인팅 도중에 생신 연산 시 UCC를 고치는 과정과 페이지 내용을 고치는 과정이 원자적으로 발생할 수 있도록 일종의 잠금을 잡아주고 상기 작업을 하게된다.

<35> 도5, 도6, 도7은 퍼지 체크포인팅을 하는 경우에 디퍼런셜 로깅을 적용하기 위한 알고리즘에 관한 도면으로, 도5는 생신 알고리즘, 도6은 퍼지 체크포인팅 알고리즘, 도7은 재시작 알고리즘이다. 상기 생신 알고리즘, 퍼지 체크포인팅 알고리즘, 재시작 알고리즘의 사용을 통해 퍼지 체크포인팅 하에서 디퍼런셜 로깅을 사용할 수 있다.

<36> 도5는 퍼지 체크포인팅 하에서 디퍼런셜 로깅을 적용하기 위한 페이지 이미지의 생

신을 하는 방법을 설명한 것이다. 우선 체크 포인트 플래그가 셋 상태인지 아닌지를 먼저 판단한다. 체크포인팅이 이루어지지 않는 경우라면 일관성 체크포인팅과 다를 바가 없으므로 곧바로 페이지의 이미지를 갱신한다(59). 하지만 체크포인팅중이라면 상기한 바와 같이 갱신 연산 시 UCC를 고치는 과정과 페이지 내용을 고치는 과정이 원자적으로 발생할 수 있도록 갱신하려는 페이지에 대한 잠금 획득을 우선적으로 한다(53). 잠금 상태에서 페이지 UCC를 증가하는 작업(55)과 페이지 이미지를 페이지 이미지를 갱신하는 작업(56)을 한다. 페이지 이미지 갱신이 이루어지면 잠금을 해제(57)하고 갱신 로그를 로그 버퍼에 삽입한다(58).

<37>        도6은 디퍼런셜 로깅을 적용하기 위한 퍼지 체크포인팅 과정을 설명한 것이다. 일반적인 퍼지 체크포인팅과 같은 과정을 거치게 되나 더티페이지에 대한 페이지 이미지를 비동기적으로 백업 디스크에 반영하는 경우에 문제가 있다. 여기서 더티 페이지란 가장 최근에 성공한 체크포인팅 이후에 갱신이 이루어진 적이 있는 페이지를 말한다. 이러한 더티 페이지를 백업 디스크에 기록하는 과정에서 사용자에 의한 다른 트랜잭션이나 연산들의 수행이 일어나는 경우 페이지 번호와 페이지 내용사이에 일관성을 잃을 수가 있다. 따라서 이 경우에도 도5에서와 마찬가지로 우선적으로 페이지에 대한 잠금을 획득하는 과정(61)을 거친다. 잠금을 획득한 상태에서 페이지 이미지를 비동기적으로 백업 디스크에 기록을 하고(62) 잠금을 해제한다(63). 잠금을 해제한 이후 더티플래그를 언셋하고, 페이지 UCC를 언셋한 이후에는 기존의 퍼지 체크포인팅과 같은 과정을 거치게 된다.

<38>        도7은 퍼지 체크포인팅 하에서 디퍼런셜 로깅을 적용하기 위한 재시작 과정에 대한 흐름도이다. 단 여기서의 재시작 방법은 후술하는 단일 경로 재시작 기법하고는 다른 것으로서 일반적인 종래의 재시작 기법인 이중 경로 재시작 기법이다. 따라서 재시작시의

로그 스캔은 체크포인트 시작로그부터 정방향으로 이루어진다. 루저(loser) 트랜잭션들을 구분해내기 위해 ATT(Active Transaction Table)를 TID와 LastLSN의 두 개의 분야로 유지한다. 트랜잭션 시작 로그인 경우에는 TID를 ATT에 삽입하며(705), 트랜잭션 완료/취소 로그인 경우에는 ATT에서 TID를 삭제하고(706) 다음 로그를 읽는 단계로 넘어간다. 갱신이 있는 갱신로그의 경우에는 ATT의 LastLSN을 갱신하고(707) 체크포인트 종료 로그를 읽은 후인지 여부를 판단(708)하여 갱신 내용에 대해 리두를 하는 것이 일반적이다. 단 본 발명에서는 퍼지 체크포인팅 하에서 사후이미지에 디퍼런셜 로깅을 적용하는 부작용을 막기 위해 이미 설정한 로그 UCC와 페이지 UCC를 비교하여 로그 UCC가 페이지 UCC보다 큰 경우에만 갱신로그에 대해 리두를 수행하며, 그렇지 않은 경우에는 다음 로그를 읽는 단계로 넘어간다. 정방향으로 로그를 읽어가면서 로그의 끝까지 도달을 하게 되면 ATT에 남아 있는 TID 즉 루저 트랜잭션에 대해 역방향으로 진행하며 언두를 한다 (711). 이와 같은 과정을 거치게 되면 디퍼런셜 로깅을 적용하여 재시작 과정에서 데이터 복구가 이루어진다.

<39>      도8은 디스크 상에 원본 데이터를 관리하는 디스크 기반 디스크 기반 관리 시스템 (Data Base Management System, 이하 'DBMS'라 함)에서의 트랜잭션 프로세싱 시스템의 구조이다. 디스크 기반 DBMS에서는 체크포인팅 도중이 아닐 때에도 메모리에 올라온 페이지들이 디스크에 반영되기 때문에 항상 로그의 일련번호를 각 페이지에 기록해두어야 한다. 또한 퍼지 체크포인팅의 경우와 마찬가지로 페이지의 내용 변경과 일련번호를 페이지에 기록해 주는 과정은 원자적으로 발생할 수 있도록 잠금이 필요하다. 디스크 기반 DBMS에서의 또 다른 문제는 디스크 쓰기 연산 도중에 시스템이 붕괴되는 경우의 복구 방법이다. 이때에는 디스크에 저장된 페이지의 이미지가 일관성이 없어지기 때문에 디퍼런

설 로깅을 적용할 경우 정확한 이미지로의 복구가 불가능해진다. 주 메모리 DBMS에서 퍼지 체크포인팅 도중 시스템 붕괴가 발생하면 그 이전의 백업 이미지로부터 시스템 복구를 시작하면 되기 때문에 이러한 문제가 발생하지 않는다. 본 발명에서는 디스크 기반 DBMS에서 디퍼런셜 로깅을 적용할 경우, 이러한 문제를 해결하기 위해 소량의 바槐발성 메모리를 디스크 버퍼(81)로 사용하여 디스크 쓰기 연산을 항상 원자적으로 발생하도록 하여준다.

<40>      도9는 디스크 기반 DBMS에서 페이지의 이미지를 생성하는 과정을 설명한 것이다. 전술한 바와 같이 페이지 이미지의 생성과 일련번호를 페이지에 기록해주는 과정은 원자적으로 발생하여야 하므로 잠금 작업을 우선적으로 한다(91). 잠금 상태에서 로그 LSN을 페이지 LSN에 반영하는 단계(92)를 거치고, 페이지 이미지를 생성한다(93). 이미지 생성 후에는 잠금을 해제하는 단계(94)를 거치고 생성 로그를 로그 버퍼에 삽입한다(95).

<41>      도10은 디스크 기반 DBMS에서 디퍼런셜 로깅을 적용한 재시작 과정을 설명한 것이다.

<42>      재시작 과정은 도7의 퍼지 체크포인팅 하에서의 재시작 과정과 대부분에 있어 동일하다. 도7에서와 같이 체크 포인트 시작부터 로그를 읽어들이고 로그 스캔을 하는 방향은 정방향으로 한다. 리두를 하지 않는 뿐만 아니라 트랜잭션을 분리하는 방법도 도7에서와 마찬가지로 로그의 타입을 판단한 후에 트랜잭션 시작로그인 경우에는 TID를 ATT에 삽입하며, 트랜잭션 완료/취소 로그인 경우에는 ATT에서 TID를 삭제하여 ATT에 TID가 남아있는지 여부로 판단한다. 생성이 있는 로그에 대해서는 ATT의 LastLSN을 생성하고, 이미 생성 연산이 반영된 페이지에 그 생성연산에 의해

생성된 로그를 또다시 적용하는 것을 막기 위해 로그 LSN과 페이지 LSN을 비교하여 로그 LSN이 페이지 LSN보다 큰 경우에만 리두를 거치게 된다(101). 그렇지 않은 경우 다음 로그를 읽는 단계로 넘어간다. 로그의 끝까지 도달하게 되면 ATT에 남아있는 TID에 대해 리두하고는 반대방향으로 언두를 한다.

<43> 기존의 물리적 로깅 방법에서는 교환성과 결합성이 없으므로 재시작 과정 시 로그의 생성순서순으로 리두를 하여야 하고, 생성 순서의 역순으로 언두를 하여야 한다. 즉 최소한 두 번 이상의 로그 스캔을 하여야만 재시작 과정이 완료된다. 그러나 디퍼런셜 로깅을 쓸 경우, 리두 연산과 언두 연산간의 구분이 필요 없기 때문에 한번의 로그 스캔만으로 재시작 과정이 완료되는 단일 경로 재시작 기법이 가능하다.

<44> 도12는 주 메모리 상에 원본 데이터를 관리하며 일관성 체크포인팅을 사용하는 시스템의 단일 경로 재시작 알고리즘이다. 본 발명에서는 로그 스캔을 하는 순서를 로그 생성 순서의 반대로 즉 로그 디스크 끝에 있는 로그부터 읽어들인다. 기존의 방식은 리두가 로그의 생성순서에 의해 이루어져야 하는 제약 때문에 어떤 트랜잭션이 완료 트랜잭션인지, 루저 트랜잭션인지 알 수가 없으며 결국 모든 트랜잭션에 대해서 일단 리두를 해야한다. 그 후에 로그 생성 순서의 역순으로 로그 스캔을 하면서 트랜잭션의 종류를 판별하고 언두를 하게 된다. 그러나 디퍼런셜 로깅을 사용할 경우 리두를 생성순서 순으로 해야 한다는 제약이 없기 때문에 완료 트랜잭션인지 루저 트랜잭션인지 판별하면서 리두를 할 수 있도록 로그 생성 순서의 반대로 로그 스캔을 한다. 로그 스캔을 하면서 완료 로그가 있는 트랜잭션에 대해서는 리두를 하여주고 루저 트랜잭션에 대해서는 백업 데이터 베이스에 기록이 되어있지 않는 이상 아무런 조치도 취하지 않고 넘어간다.

<45> 이를 도12를 참조하여 자세히 설명하면 다음과 같다. 도11에서 CTT(committed

transaction table)은 완료 로그가 있는 트랜잭션의 집합을 말하며 RTT(rolled-back transaction table)는 루저 트랜잭션에 대한 집합이다. 도12에서 역방향으로 로그를 스캔하면서 리두(125)를 하는 경우는 로그 타입을 판단(122) 시 갱신로그이면서 CTT에 TID가 있는 경우이다. 로그 타입을 판단 시 트랜잭션 시작 로그인 경우에는 CTT에서 TID를 삭제하며(123), 트랜잭션 완료로그인 경우에는 CTT에 TID를 삽입(124)하고 다음 로그를 읽어들인다. 체크 포인트 종료로그까지 도달한 후에는 체크포인트 종료로그에 있는 TID들로 RTT를 구성하고 RTT에 있는 TID들 중 CTT에도 있는 TID는 삭제한다. 그 후에는 RTT에 나아 있는 TID에 대해 리두와 같은 방향으로 계속 진행하며 언두를 한다(127). 언두가 필요한 트랜잭션 즉 도12에 의하면 RTT와 CTT에 공통적으로 존재하는 TID를 삭제하고 남은 RTT에 존재하는 TID는, 취소된 트랜잭션중 시작이 가장 최근에 성공한 체크포인트 이전에 시작된 트랜잭션이다. 이 경우에 언두가 필요한 이유는 상기 트랜잭션에 대한 결과는 이미 디스크에 반영이 되었기 때문에 취소할 필요가 있기 때문이다.

<46>        도11은 퍼지 체크포인팅 하에서 디퍼런셜 로깅에 의한 단일경로 재시작 알고리즘이다.

<47>        도11에서 알 수 있듯이, 퍼지 체크포인팅의 경우에 로그 스캔을 역방향으로 하면서 체크포인트 종료로그까지 도달할 때에는 일관성 체크포인팅의 경우와 동일한 것을 알 수 있다. 하지만 체크 포인트 종료로그부터 체크포인트 시작로그까지 도달하는 때에는 문제가 생긴다. 이는 체크포인팅 도중에도 갱신로그가 발생하기 때문이다.

<48>        이에 대한 처리를 도11을 참조하여 설명하면 다음과 같다. 우선 로그 타입을 판단하여 갱신로그가 아닌 경우에는 다음과 같이 처리한다. 트랜잭션 시작 로그의 경우에는 CTT와 RTT에서 TID를 삭제하고, 트랜잭션 완료로그인 경우에는 CTT에 TID를 삽입하며,

트랜잭션 취소로그인 경우에는 RTT에 TID를 삽입한다. 개신로그이면서 완료 트랜잭션인 경우에는 해당 개신로그를 발생시킨 개신 연산의 영향이 디스크에 저장된 백업 데이터베이스에 반영이 되었는가에 따라 처리가 달라진다. 이미 그 영향이 반영이 되었다면 리두를 할 필요가 없으며, 개신 연산에 의한 영향이 되지 않았다면 리두를 하여야 한다. 디스크에 반영이 되었는지의 여부는 로그 UCC와 페이지 UCC를 비교하여 판단하며, 로그 UCC가 페이지 UCC보다 크거나 같은 경우에는 디스크에 반영이 된 것이고, 그렇지 않은 경우에는 디스크에 반영이 되지 않은 것이므로 리두를 하여야 한다(111). 개신로그이면서 루저 트랜잭션인 경우에도 개신 연산의 영향이 디스크에 반영이 되었는가에 따라 처리가 달라진다. 개신 연산의 영향이 디스크에 반영이 되었다면 이를 취소시켜 주어야 하므로 언두를 하여야 하며, 디스크에 반영이 되지 않았다면 추가적인 연산을 할 필요는 없다(112). 루저 트랜잭션이 디스크에 반영되었는지의 여부는 완료 트랜잭션의 경우와 마찬가지로 로그 UCC와 페이지 UCC를 비교해서 판단한다. 로그 UCC가 페이지 UCC보다 작으면 디스크에 반영이 된 것이므로 언두 하여야 하고, 반대에 경우에는 그렇지 않다. 체크 포인트 시작 로그 이후에는 일관성 체크포인팅의 경우와 같다. 이경우에 언두가 필요한 경우는 일관성 체크포인팅과 마찬가지로 취소된 트랜잭션 중 시작이 가장 최근에 성공한 체크포인트 이전에 시작된 트랜잭션이다.

<49> 일관성 체크포인팅에 의해 단일 경로 재시작을 하는 경우나, 퍼지 체크포인팅에 의해 단일 경로 재시작을 하는 경우 모두, 정상 상태에서 트랜잭션이 취소되면 이미 수행했던 연산에 대한 CLR(compensation log record)를 생성할 필요가 없으며 취소된 트랜잭션의 로그들을 로그 디스크에 반영할 필요가 없다. 따라서, 트랜잭션 취소율이 높은 환경에서는 로깅 및 재시작 성능 향상에 기여할 수 있다.

<50> 도13은 디스크 상에 원본 데이터를 관리하는 시스템의 단일 경로 재시작 알고리즘이다.

<51> 도13에서 알 수 있듯이 재시작 기법은 일관성 체크포인팅에서의 단일 경로 재시작 기법과 대부분 유사하다. 다만 디스크 기반 DBMS에서는 체크포인팅~여부를 막론하고 갱신로그를 생성하므로 이에 대한 처리가 문제된다.

<52> 갱신 로그이면서 완료트랜잭션인 경우, 갱신 연산의 영향이 디스크에 저장될 때 업데이트 베이스에 반영이 되었는가에 따라 처리가 달라지는데 이미 그 영향이 반영이 되었다면 리두를 할 필요가 없으며, 갱신 연산에 의한 영향이 반영이 되지 않았다면 리두를 하여야 한다. 이에 대한 판단은 로그 LSN과 페이지 LSN의 비교를 통해서 하며, 로그 LSN이 페이지 LSN보다 크면 리두를 하고 그렇지 않으면 다음 로그를 읽는 단계로 넘어간다 (131).

<53> 갱신 로그이면서 루저 트랜잭션인 경우, 갱신 연산의 영향이 디스크에 반영이 되었다면 이를 취소시켜 주어야 하므로 언두를 하여야 하고, 디스크에 반영이 되지 않았다면 추가적인 연산을 할 필요는 없다. 이에 대한 판단 역시 로그 LSN과 페이지 LSN의 비교를 통해서 한다. 로그 LSN이 페이지 LSN보다 작거나 같으면 언두를 하게 되고 그렇지 않으면 다음 로그를 읽는 단계로 넘어단다(132).

<54> 디스크 기반 DBMS에서의 단일 경로 재시작 기법은 트랜잭션이 취소되더라도 그 트랜잭션의 로그는 로그 디스크에 반영되어야 하지만, CLR은 생성할 필요가 없다.

<55> 디퍼런셜 로깅에서는 로그의 생성 순서와 상관없이 임의의 순서대로 리두나 언두 되더라도 정확한 시스템 상태로의 복구가 가능하기 때문에 병렬적 재시작 기법에 효과적

으로 적용될 수 있다.

<56>      도14는 복수의 로그 디스크가 존재하는 시스템에서 각 로그 디스크마다 별도의 로그 관리자를 두고서 트랜잭션 별로 로그를 분배하는 병렬적 로깅 시스템의 구조이다.

<57>      일단, 트랜잭션이 시작되면, 당시에 가장 부하가 적게 걸린 로그 관리자와 LogMgr ID를 해당 트랜잭션에 할당하여 그 트랜잭션에서 생성되는 로그는 항상 그 로그 디스크(141)에만 저장되도록 한다. 그리고, 체크포인팅 시작 로그나 종료 로그는 모든 로그 디스크에 쓰이도록 한다.

<58>      도 15는 도 14와 같이 로깅을 한 시스템에서 병렬적으로 재시작하는 구조를 나타낸 도면이다. 재시작 시에는 각 로그 디스크(151)마다 별도의 쓰레드(152)와 별도의 트랜잭션 테이블(153)을 할당하여 각 로그 디스크들이 독립적으로 재시작될 수 있도록 한다. 또한 ATT나 RTT 또는 CTT와 같이 재시작 시 필요한 트랜잭션 테이블은 별도로 관리한다.

<59>      물론, 로그 디스크가 하나만 존재하는 경우에도 도16과 도17과 같은 방법을 사용하여 병렬적 재시작이 가능하다. 도16은 로그를 읽는 하나의 쓰레드(162), 로그를 분배하는 하나의 쓰레드(163), 자신의 재시작 큐(164)에 도착한 로그에 대한 리두나 언두를 수행하는 여러 개의 쓰레드(165)로 구성된다. 도17은 로그를 읽는 하나의 쓰레드(172)와 로그 버퍼(173)에서 하나의 로그를 읽어 이에 대한 리두나 언두를 수행하는 여러 개의 리두와 언두 쓰레드(174)로 구성된다.

<60>      주 메모리에 원본 데이터를 관리하는 시스템의 재시작 과정은, 백업 데이터를 원본 데이터에 반영하는 과정과 로그를 원본 데이터에 반영하는 과정으로 구성되는데, 디퍼런

설 로깅을 사용할 경우 이 두 과정을 병렬적으로 수행할 수 있다.

<61> 도18은 복수의 로그 디스크와 복수의 백업 디스크를 이용한 병렬적 재시작 구조에 관한 도면이다. 도18에서 알 수 있듯이 백업 데이터를 원본 데이터에 반영하는 과정은 백업 디스크에서 백업 데이터를 읽어 주메모리에 올리는 과정(BDR)(181)과 백업 데이터 캐노를 원본 데이터에 복사하는 과정(182)으로 구성되며, 로그를 원본 데이터에 반영하는 과정은 로그 디스크에서 로그를 읽어 주 메모리로 올리는 과정(LR)(183)과 로그를 원본 데이터에 리두나 언두하는 과정(LP)(184)으로 구성된다. 도 18에 의하면 이러한 각 과정들은 각 백업 디스크마다 독립적으로 수행하며, 로그 디스크(!86)의 수가 여러 개이면 BDR과 BDP를 각 백업 디스크마다 독립적으로 수행하며, 또한 도19와 같은 알고리즘을 사용하게 되면 백업 데이터를 주 메모리의 원본 데이터에 반영하는 과정과 로그를 주 메모리의 원본 데이터에 반영하는 과정을 병렬적으로 수행할 수 있다.

### 【발명의 효과】

<62> 본 발명에서 새롭게 제안하는 기법인 디퍼런셜 로깅을 이용할 경우 로깅 성능과 재시작 성능의 현격한 향상을 기대할 수 있다. 디퍼런셜 로깅이 그러한 성능 향상을 가능케 하는 가장 큰 이유는 디퍼런셜 로깅의 교환성과 결합성에 있다. 복수의 로그 디스크와 다중 프로세서가 존재하는 환경에서 기존의 로깅 방식을 사용할 경우 정확한 회복을 위해서는 로깅 시 복수의 로그 디스크를 충분히 활용할 수 없거나 재시작 시 로그들의 생성 순서 파악을 위한 오버헤드가 존재한다. 그러나, 디퍼런셜 로깅을 사용할 경우 로그들을 임의로 로그 디스크에 분배할 수 있으므로 복수의 로그 디스크를 충분히 활용할 수 있을 뿐만 아니라 재시작 시에도 로그 생성 순서 파악을 위한 오버헤드가 없다. 그리

고, 재시작 시 각 로그 디스크 간의 병렬적인 재시작이 가능할 뿐만 아니라 로그를 처리하는 과정과 백업 데이터베이스를 메모리로 올리는 과정을 병렬적으로 수행할 수 있기 때문에 재시작 시간이 크게 줄어든다.

<63> 또한, 디퍼런셜 로깅에서는 리두와 언두간의 구분이 없으므로 단일 경로 재시작 기법이 가능하다. 이러한 재시작 기법을 쓸 경우 CLR을 만들 필요가 없고, 퍼지 체크포인트 중이 아니라면 트랜잭션 수행 중 취소된 것들에 대해서는 로그를 디스크에 저장할 필요가 없어진다. 따라서, 단일 경로 재시작 기법은 재시작 성능의 향상 뿐만 아니라, 트랜잭션 취소율이 높은 환경에서 로깅 성능의 향상에도 기여할 수 있다.

<64> 디퍼런셜 로깅은 변경 전 이미지와 변경 후 이미지를 모두 저장하는 물리적 로깅과는 달리 이들 간의 차이만을 저장하므로 물리적 로깅에 비해 로그량을 절반 가까이 줄일 수 있다. 또한, 디퍼런셜 로그는 변경된 부분만 비트 1로 나타나기 때문에 RLE(Run Length Encoding)를 사용하면 빠른 속도로 높은 압축률을 얻을 수 있다. 이러한 로그량의 감소는 로깅 성능과 재시작 성능의 향상에 기여할 수 있다.

<65> 이상과 같은 디퍼런셜 로깅의 발명은 전자 상거래 서버, 공정 제어, 통신 시스템 등과 같이 고성능을 요하는 분야에서 주 메모리 데이터에 대한 로깅 기법으로 활용될 경우 시스템의 로깅 및 재시작 성능을 크게 향상시킬 수 있다.

1020000031166

2000/12/1

국내시작

**【특허청구범위】****【청구항 1】**

트랜잭션 단위로 원자성과 지속성을 보장해야 하는 시스템을 위한 로깅 방식에 있어서, 로그 레코드의 본체는 변경된 데이터의 변경 전 이미지와 변경 후 이미지간의 XOR 결과로 이루어지며 리두 연산 또는 언두 연산은 상기 로그 레코드의 본체와 상기 로그 레코드가 생성되었던 위치에 저장되어 있는 데이터간의 XOR 연산을 통해 이루어지는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 2】**

제 1항에 있어서, 상기 로그레코드는 로그의 생성 순서를 나타내며 주로 로그 디스크의 물리적 주소값이 저장되는 LSN(11), 상기 로그 레코드를 만들어낸 트랜잭션에 의해 가장 최근에 만들어진 로그 레코드의 아이디인 PrevLSN(12), 상기 로그 레코드를 생성한 트랜잭션의 아이디 정보를 나타내는 TID(13), 상기 로그 레코드의 타입을 나타내는 정보인 type(14), 퍼지 체크포인트와 함께 사용 시 로그와 변경된 페이지 사이의 관계를 나타내는 UCC(15), 변경된 페이지의 위치를 나타내는 PageID(16), 변경된 슬롯의 위치를 나타내는 Slot ID(17), 변경된 슬롯의 길이를 나타내는 Size(18)를 포함하는 로그 헤더와 상기 로그 본체를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 3】**

제 1항에 있어서, 상기 디퍼런셜 로깅에 의한 회복방법을 퍼지 체크포인팅에 적용

하기 위하여 UCC를 고치는 과정과 페이지 내용을 고치는 과정이 원자적으로 이루어질 수 있도록 갱신 연산을 수행하는 단계; 페이지 이미지의 일관성을 유지하면서 퍼지 체크포인팅을 수행하는 단계; 시스템 재시작 단계를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 4】

제 3항에 있어서, 상기 갱신 연산의 수행단계는 갱신하려는 페이지에 대한 잠금을 획득하는 단계(53); 페이지 UCC를 갱신 로그에 반영하고 페이지 UCC를 증가하는 단계(54,55); 페이지 이미지를 갱신하는 단계(56); 잠금을 해제하는 단계(57); 갱신 로그를 로그 버퍼에 삽입하는 단계(58)를 더 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 5】

제 3항에 있어서,

상기 퍼지 체크포인팅의 수행 단계는 더티 페이지가 존재할 경우, 페이지 이미지의 일관성을 유지하기 위해 페이지에 대한 잠금을 획득하고(61) 페이지 이미지를 비동기적으로 백업디스크에 기록하는 단계(62)를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법

#### 【청구항 6】

제 3항에 있어서, 상기 재시작 단계는 체크포인트 시작 로그로부터 ATT를 읽는 단계(701); 로그 타입을 판단하는 단계(704); 상기 판단한 로그 타입이 트랜잭션 시작로

그인 경우 TID를 ATT에 삽입하고(705) 트랜잭션 완료/취소 로그인 경우 ATT에서 TID를 삭제하고(706) 다음 로그를 읽어들이는 단계(702); 상기 판단한 로그 타입이 쟁신로그인 경우 ATT의 LastLSN을 쟁신하고(707) 체크 포인트 종료로그를 읽은 후이면 로그 UCC와 페이지 UCC를 비교(709)하고 그렇지 않으면 다음 로그를 읽는 단계; ~~상기~~ 비교한 로그 UCC의 값이 상기 페이지 UCC보다 크면 리두(710)를 하고 그렇지 않으면 다음 로그를 읽는 단계; 로그의 끝까지 도달한 경우 ATT에 남아있는 TID에 대해 로그 끝에서부터 로그 ~~나~~ ~~계~~를 읽어들인 반대방향으로 진행하며 연두 하는 단계(711)를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 7】

제 1항에 있어서 상기 디퍼런셜 로깅에 의한 회복방법을 디스크 기반 DBMS에 적용하기 위해 비 휘발성 메모리를 디스크 버퍼(81)로 사용하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 8】

제 1항 내지 7항 중 어느 한 항에 있어서, 상기 디스크 기반 DBMS에서 디퍼런셜 로깅에 의한 회복 방법은 로그 LSN의 페이지 반영과 페이지 이미지 쟁신이 원자적으로 발생할 수 있도록 하는 페이지 쟁신단계; 회복 연산을 위한 재시작 단계를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

### 【청구항 9】

제 8항에 있어서, 상기 페이지 갱신 단계는 갱신하려는 페이지에 대한 잠금을 회복하는 단계(91); 상기 로그 LSN을 상기 페이지 LSN에 반영하는 단계(92); 상기 페이지 이미지를 갱신하는 단계(93); 잠금을 해제하는 단계(94); 갱신로그를 로그 버퍼에 삽입하는 단계(95)를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

### 【청구항 10】

제 8항에 있어서, 상기 재시작 단계는 체크포인트 시작 로그로부터 ATT를 읽는 단계; 로그 타입을 판단하는 단계; 상기 판단한 로그 타입이 트랜잭션 시작 로그인 경우 TID를 ATT에 삽입하고, 트랜잭션 완료/취소 로그인 경우 ATT에서 TID를 삭제하고 다음 로그를 읽어들이는 단계; 상기 판단한 로그 타입이 갱신로그인 경우 ATT의 LastLSN을 갱신하고 로그 LSN과 페이지 LSN을 비교하는 단계(101); 상기 비교한 로그 LSN의 값이 상기 페이지 LSN보다 크거나 같으면 리두를 하고 그렇지 않으면 다음 로그를 읽는 단계; 로그의 끝까지 도달한 경우 ATT에 남아있는 TID에 대해 로그 끝에서부터 로그를 읽어들인 반대방향으로 진행하며 언두 하는 단계를 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

### 【청구항 11】

제 1항에 있어서,

상기 회복연산을 위한 디퍼런셜 로깅 방법은 일관성 체크 포인팅을 사용할 경우, CTT를 초기화 하고 로그 디스크 끝에 있는 로그를 읽어들이는 단계(121); 로그 타

입을 판단하는 단계(122); 상기 판단한 로그 타입이 트랜잭션 시작로그인 경우 CTT에서 TID를 삭제하고(123), 트랜잭션 완료로그인 경우 CTT에 TID를 삽입하며(124) 다음로그를 역방향으로 읽어들이는 단계(126); 상기 판단한 로그 타입이 갱신 로그인 경우 TID가 CTT에 있으면 리두(125)하고 그렇지 않으면 다음 로그를 역방향으로 읽어들이는 단계; 체크포인트 종료로그까지 도달한 경우 종료 로그에 있는 TID들로 RTT를 구성하고 상기 RTT와 CTT중 공통적으로 남아 있는 TID를 삭제한 후 RTT에 남아 있는 TID에 대해 역방향으로 언두하는 단계(127)를 포함하는 단일 경로 재시작 기법에 적용되는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

### 【청구항 12】

제 11항에 있어서,  
상기 단일 경로 재시작 기법을 퍼지 체크포인팅에 적용할 경우,  
체크 포인트 종료로그부터 체크포인트 시작로그까지의 갱신로그에 대해, TID가 CTT에 있고 로그 UCC가 페이지 UCC보다 크거나 같으면 리두하고 그렇지 않으면 다음 로그를 읽어들이는 단계(111); TID가 CTT에 없고 로그 UCC가 페이지 UCC보다 작으면 언두하고 그렇지 않으면 다음 로그를 읽어들이는 단계(112)를 더 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

### 【청구항 13】

제 11항에 있어서,  
상기 단일 경로 재시작 기법을 디스크 기반 DBMS에 적용할 경우,

생신 로그에 대해, TID가 CTT에 있고 로그 LSN이 페이지 LSN보다 크면 리두하고 그 렇지 않으면 다음 로그를 읽어들이는 단계(131); TID가 CTT에 없고 로그 UCC가 페이지 UCC보다 작으면 언두하고 그렇지 않으면 다음 로그를 읽어들이는 단계(132)를 더 포함하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 14】

제 1항에 있어서, 상기 디퍼런셜 로깅 방법은 복수의 로그 디스크를 사용할 경우 트랜잭션들을 처리하는 과정에서 생성되는 로그 레코드들을 임의의 로그 디스크로 분배 하더라도 시스템 재시작 시 로그 레코드간의 우선 순위 파악을 위한 오버헤드 없이 각 로그 디스크마다 병렬적으로 리두 도는 언두 연산을 하는 병렬적 재시작 기법에 적용되는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 15】

제 14항에 있어서, 상기 복수의 로그 디스크를 사용한 트랜잭션의 처리는 각 로그 디스크(141)마다 별도의 로그 버퍼(142)를 가진 로그 관리자(143)를 할당하고, 트랜잭션 시작 시 가장 부하가 적게 걸린 로그 디스크의 로그 관리자를 상기 트랜잭션에 할당하고 상기 트랜잭션의 로그는 해당 로그 디스크에만 저장되도록 하며, 체크 포인트 시작 로그와 체크포인트 종료로그는 모든 로그 디스크에 기록하도록 하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 16】**

제 14항에 있어서, 상기 병렬적 재시작을 수행할 수 있도록 각 로그 디스크(151)

또다른 별도의 쓰레드(152)와 별도의 트랜잭션 테이블(153)을 할당하여 상기 각 로그 디스~

크들이 독립적으로 재시작하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환

적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 17】**

제 14항에 있어서, 상기 병렬적 재시작을 수행할 수 있도록, 하나의 로그 디스크

(161), 로그를 읽는 하나의 쓰레드(162), 하나의 로그 분배 쓰레드(163), 재시작 큐

(164)에 도착한 로그에 대한 리두나 언두를 수행하는 여러 개의 쓰레드(165)를 두어 로

그가 원본 데이터에 반영되는 과정이 병렬적으로 수행되는 것을 특징으로 하는 트랜잭션

프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 18】**

제 14항에 있어서, 상기 병렬적 재시작을 수행할 수 있도록, 하나의 로그 디스크

(171), 로그를 읽는 하나의 쓰레드(172), 로그 버퍼(173)에서 하나의 로그를 읽어 이에

대한 리두나 언두를 수행하는 여러 개의 리두와 언두 쓰레드(174)를 두어 로그가 원본

데이터에 반영되는 과정이 병렬적으로 수행되는 것을 특징으로 하는 트랜잭션 프로세싱

시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

**【청구항 19】**

제 1항에 있어서, 상기 디퍼런셜 로깅 방법은 백업 데이터를 주 메모리의 원본 데

이터에 반영할 때 백업 디스크에서 백업 데이터를 읽는 과정(BDR)(181)과 읽은 데이터를

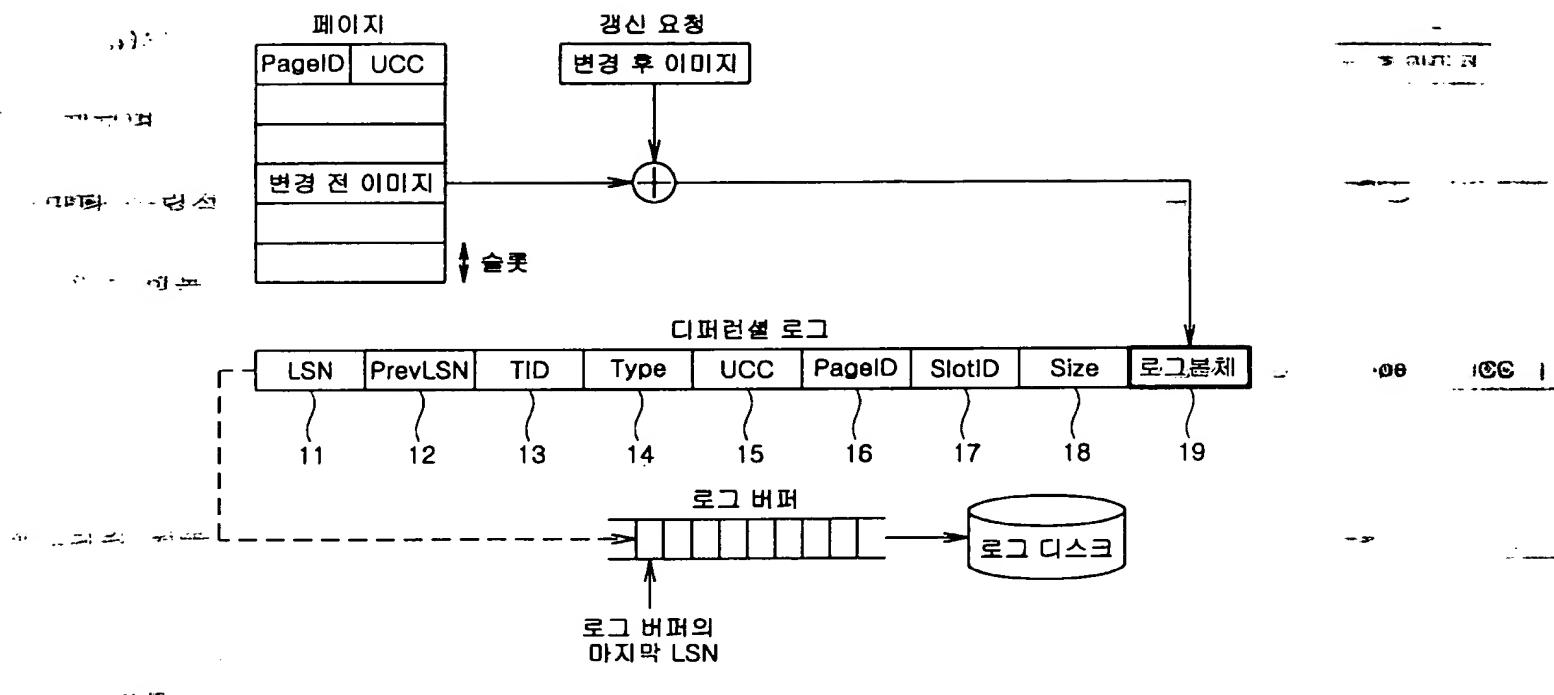
주 메모리에 반영하는 과정(BDP)(182)를 파이프 라인 방식으로 수행한 후, 로그를 주 메모리에 반영할 때 로그 디스크에서 로그를 읽는 과정(LR)(183)과 읽은 로그를 주 메모리에 반영하는 과정(LP)(184)를 파이프 라인 방식으로 수행하며, 백업 디스크(185)의 수가 여러 개인 경우에는 상기 BDR 및 BDP를 각 백업 디스크마다 독립적으로 수행하며, 그 디스크(186)의 수가 여러 개인 경우에는 상기 LR과 LP를 각 로그 디스크마다 독립적으로 수행하는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

#### 【청구항 20】

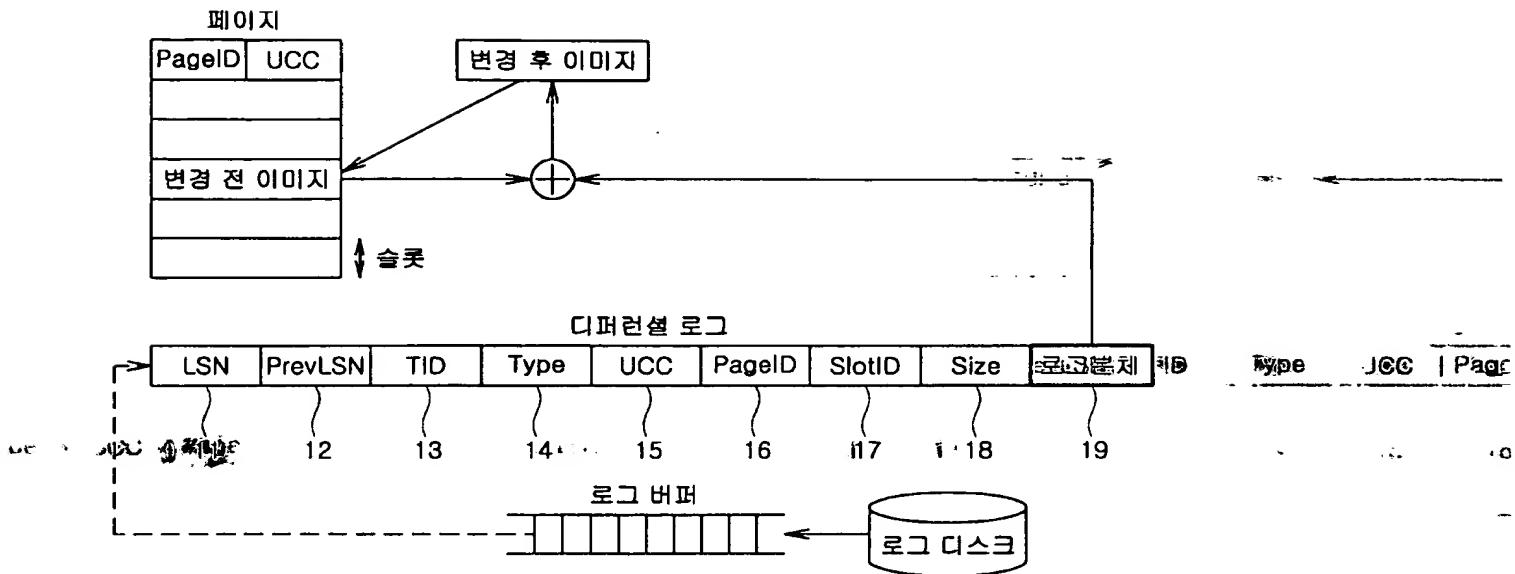
제 1항 내지 19항 중 어느 한 항에 있어서, 상기 백업 데이터를 주 메모리의 원본 데이터에 반영하는 과정과 로그를 주 메모리의 원본 데이터에 반영하는 과정은 병렬적으로 수행되는 것을 특징으로 하는 트랜잭션 프로세싱 시스템에서 교환적, 결합적 회복 연산을 위한 디퍼런셜 로깅 방법.

## 【도면】

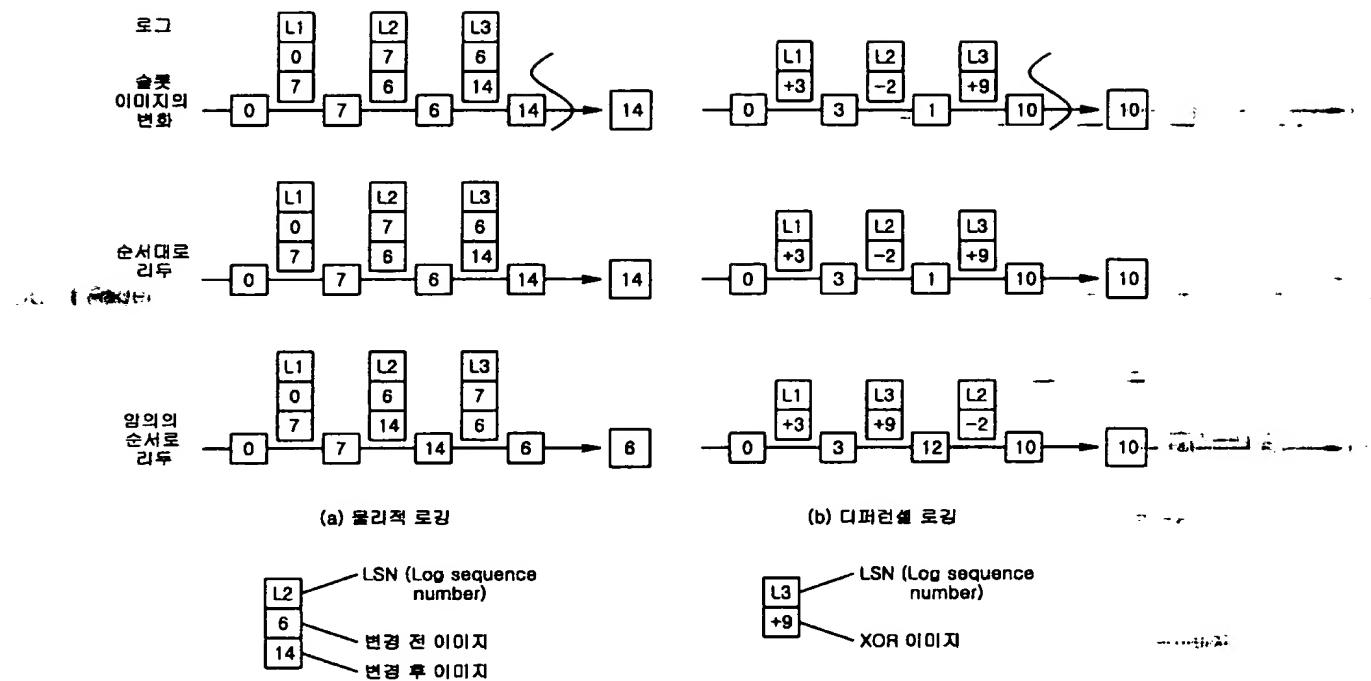
【도 1】



【도 2】



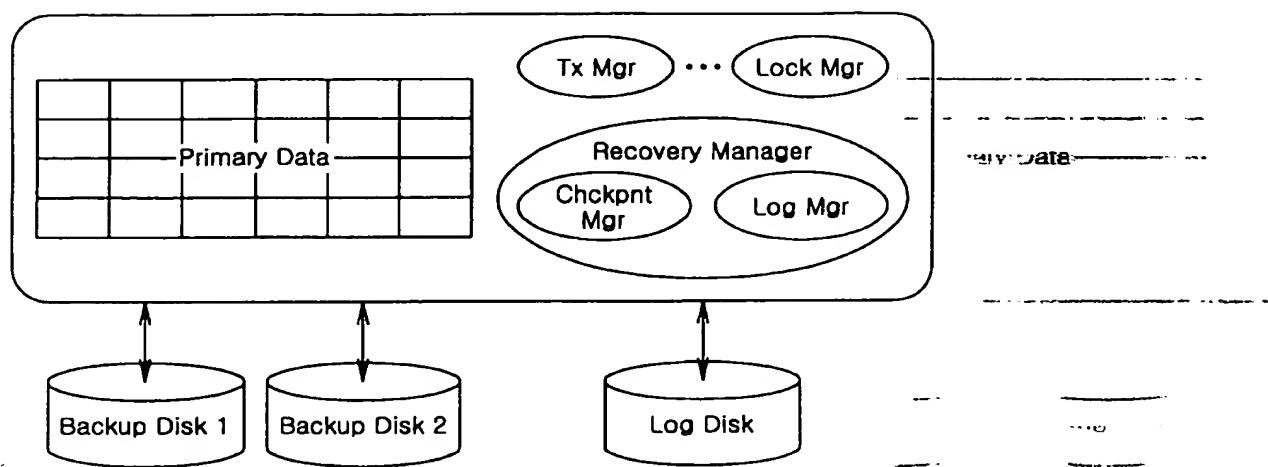
## 【도 3】



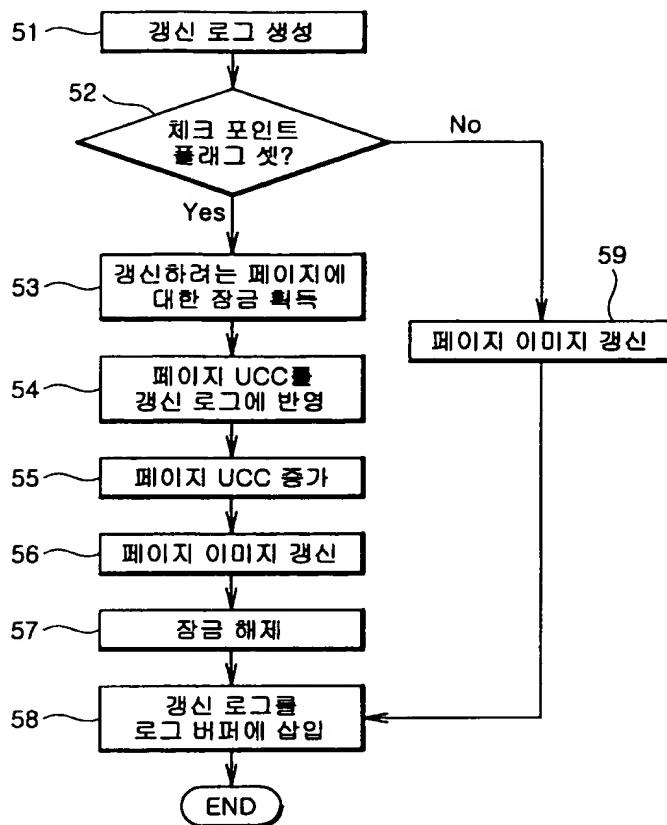
1020000031166

2000/12/1

【도 4】



【도 5】



51 ~ 캡신 로그 생성

52 ~

53 ~

54 ~

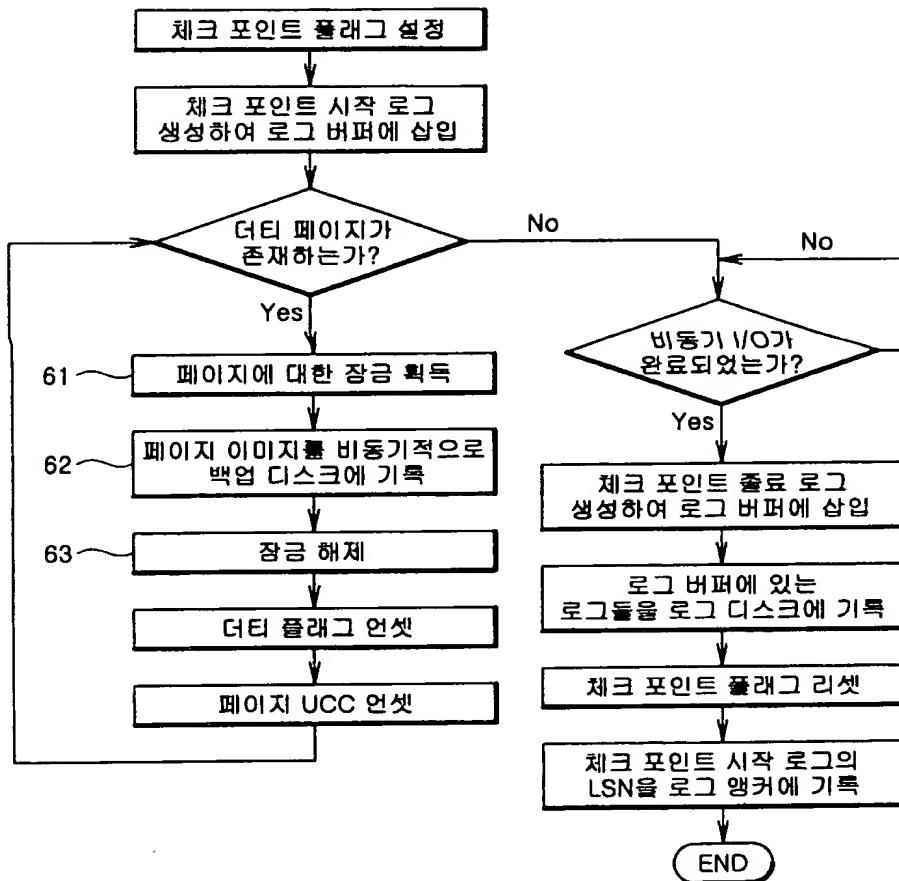
55 ~

56 ~

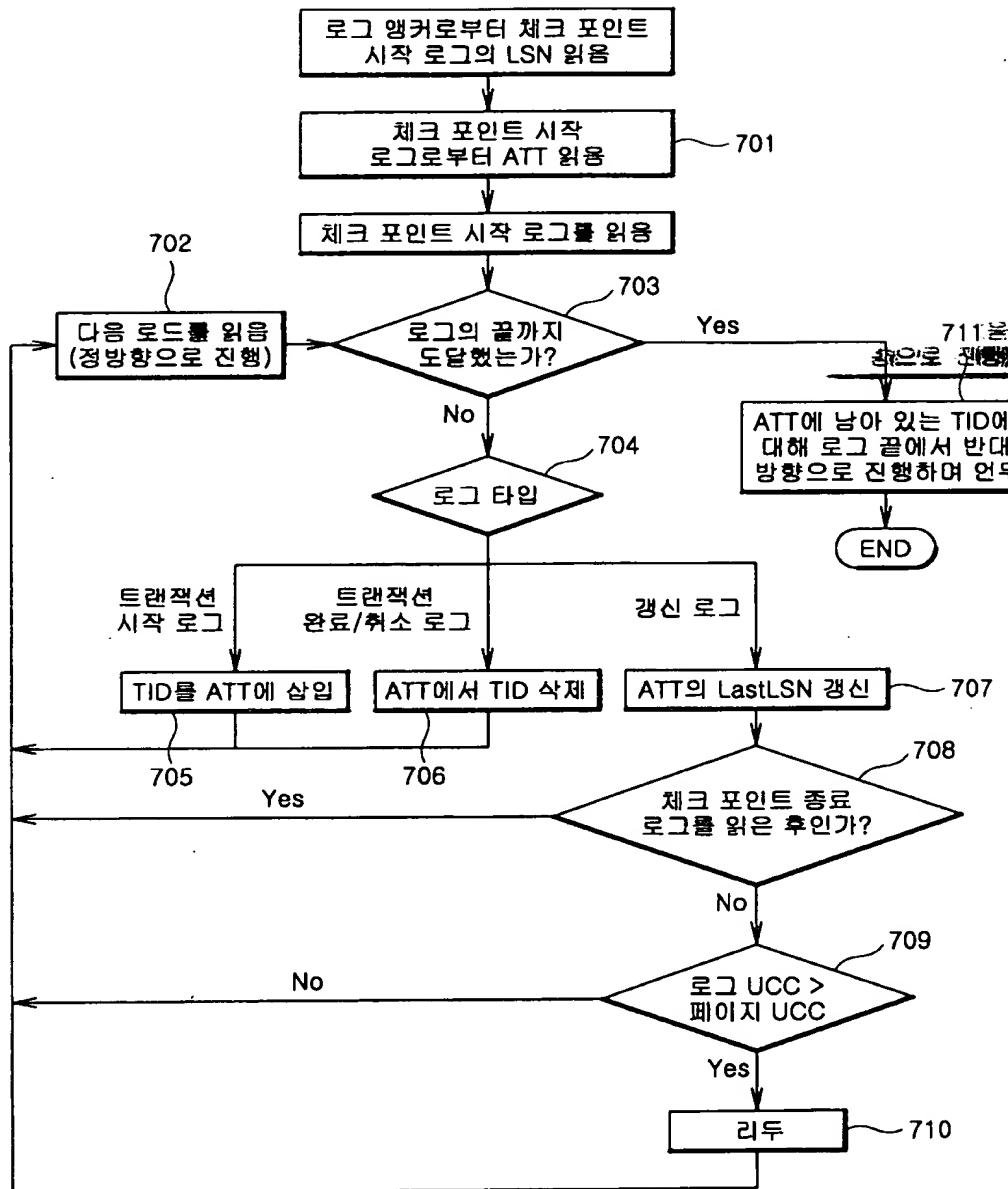
57 ~

58 ~

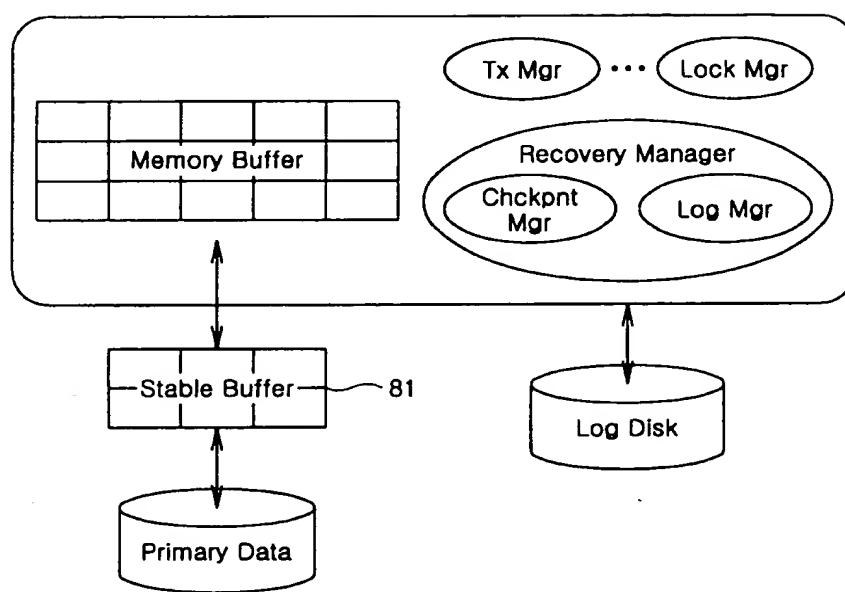
## 【도 6】



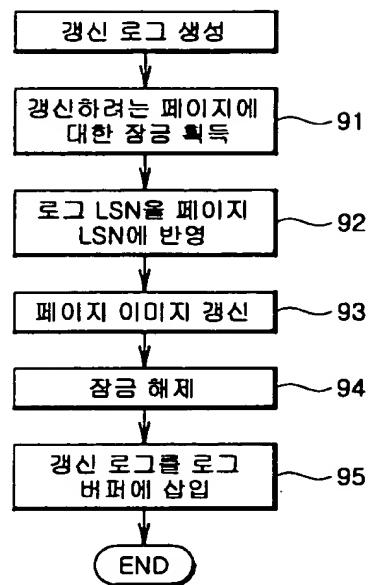
【도 7】



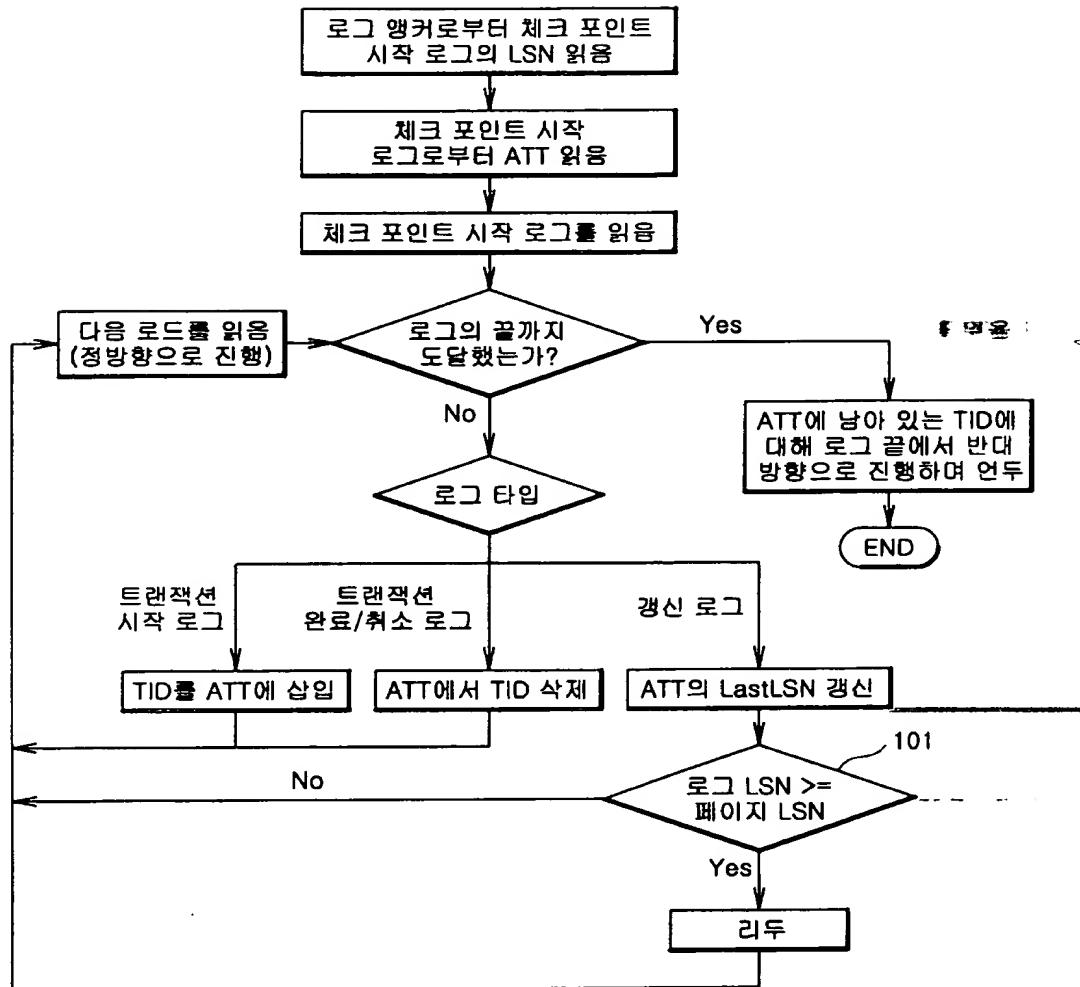
【도 8】



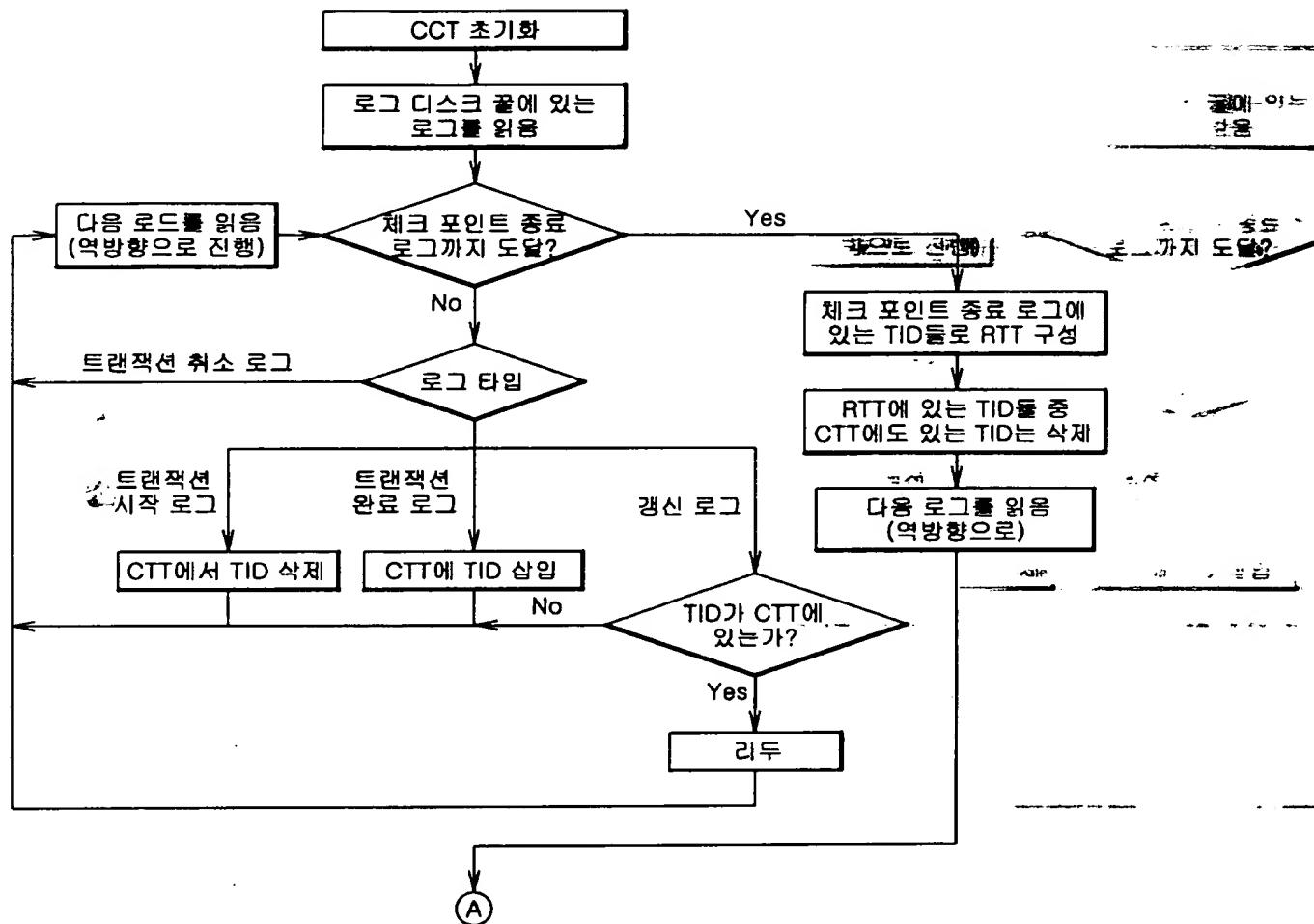
## 【도 9】



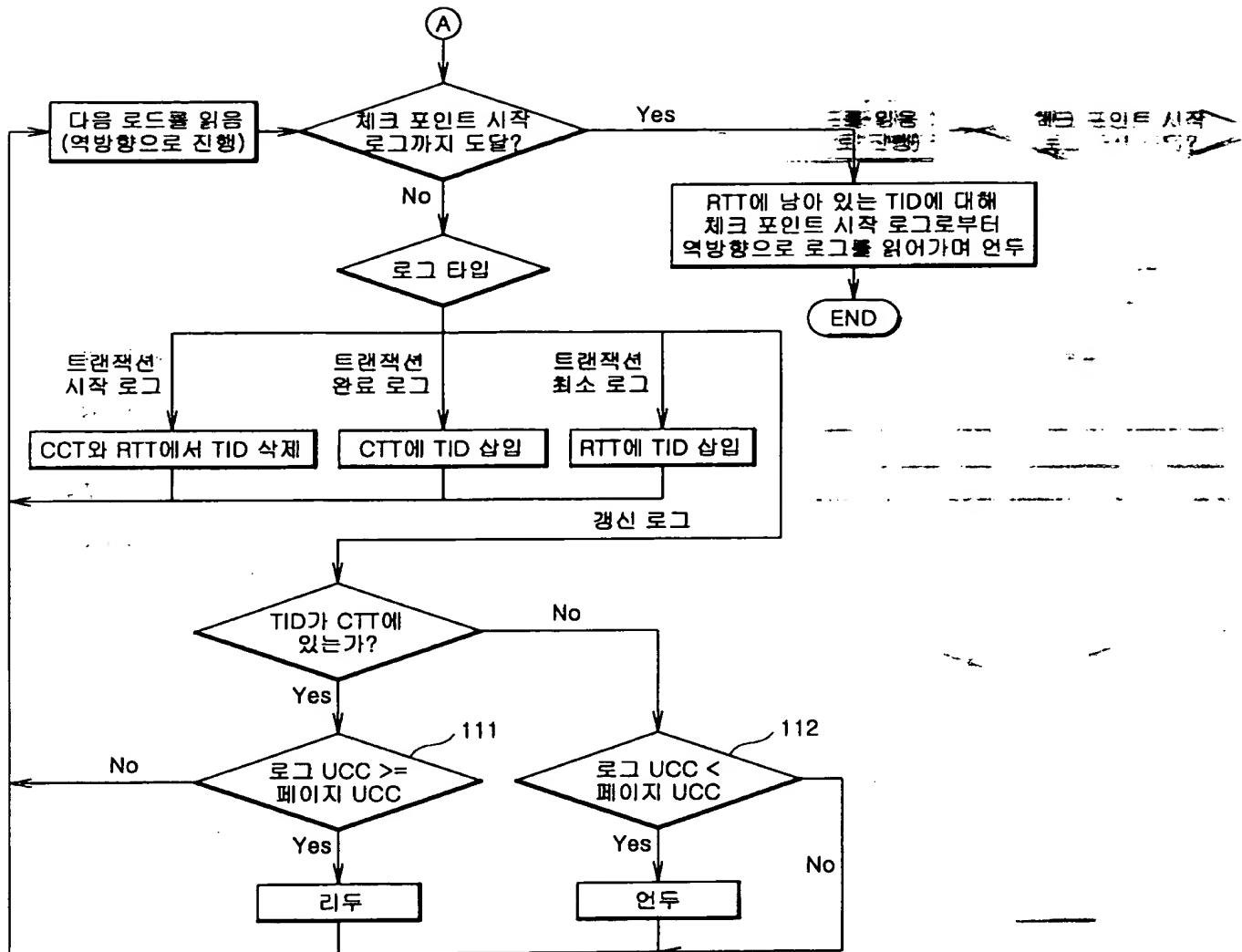
【도 10】



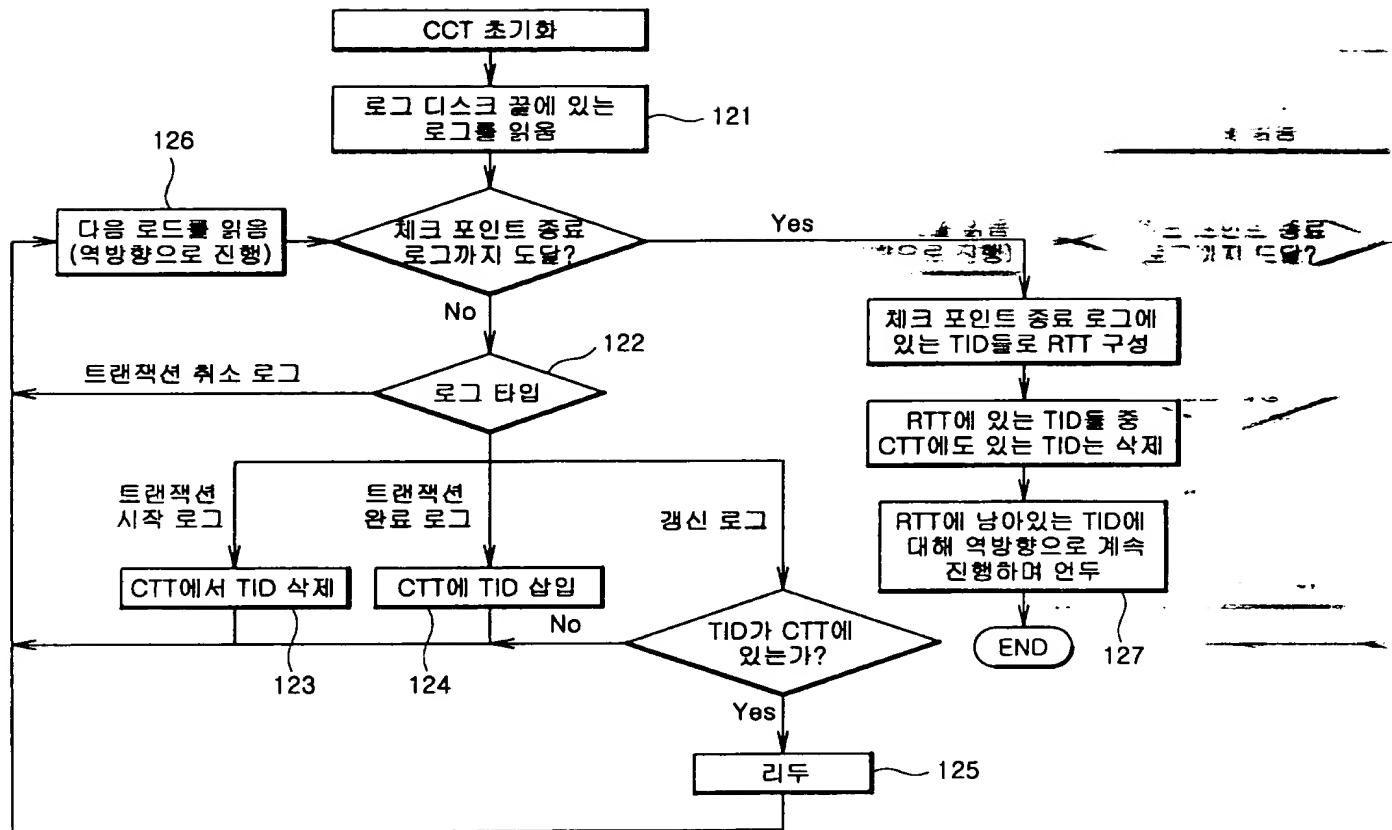
【도 11a】



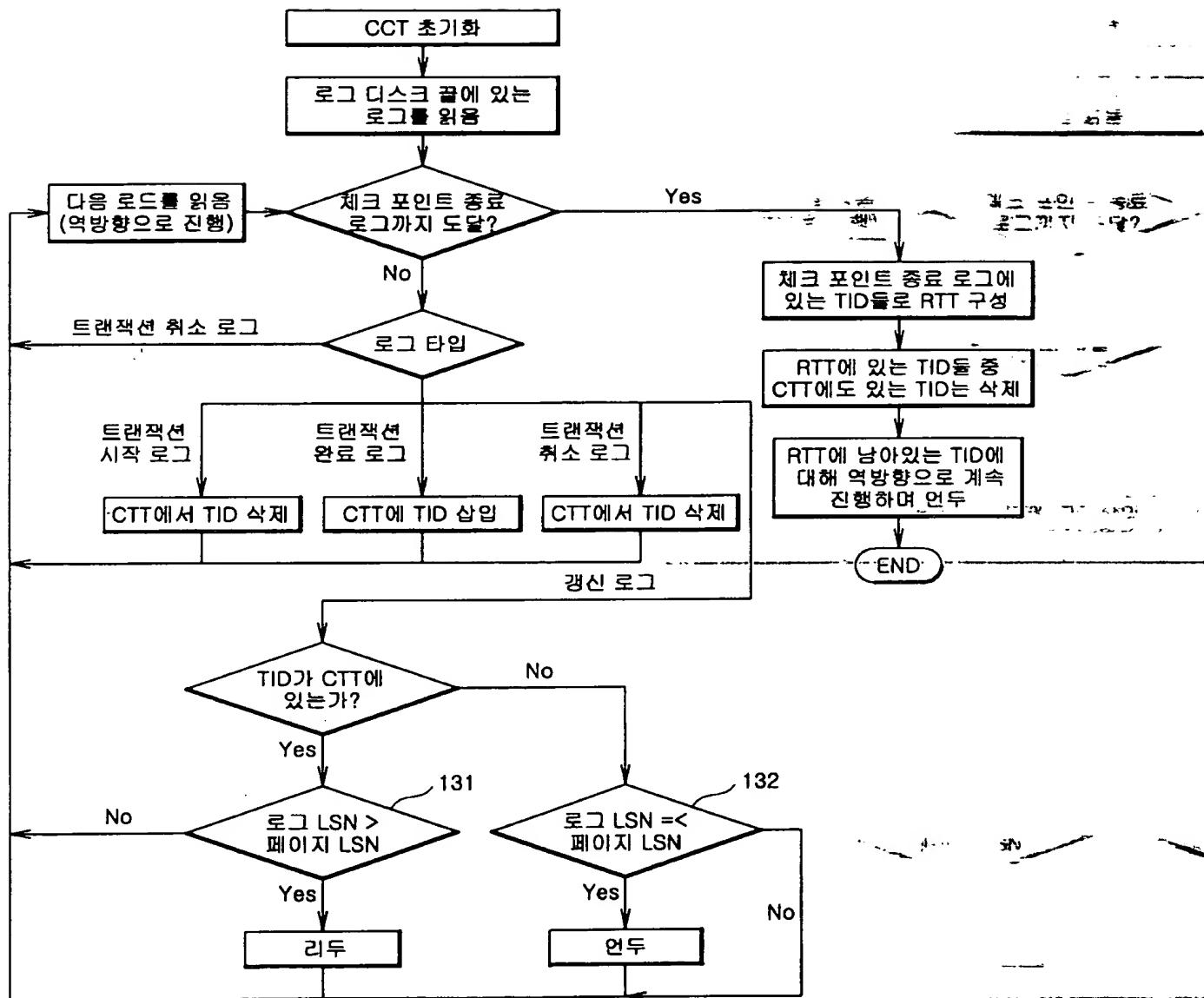
【도 11b】



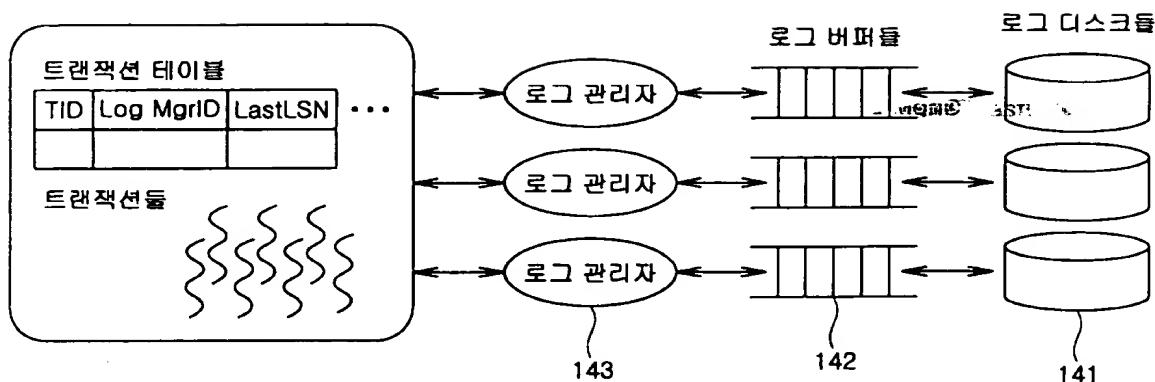
【도 12】



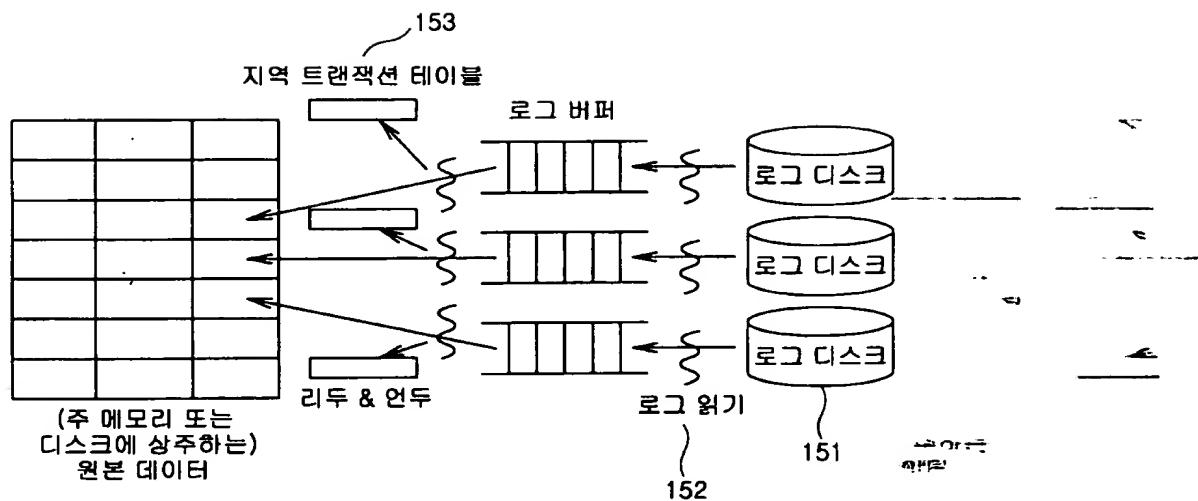
【도 13】



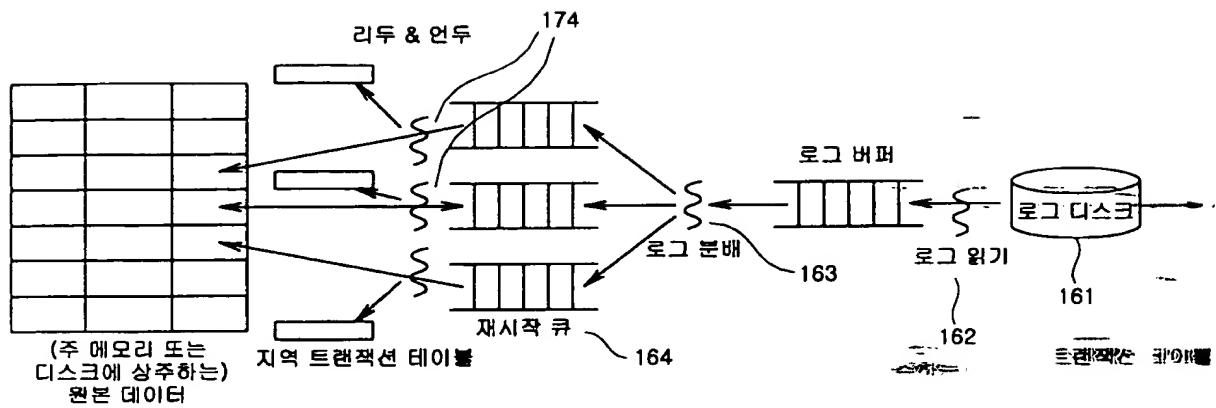
【도 14】



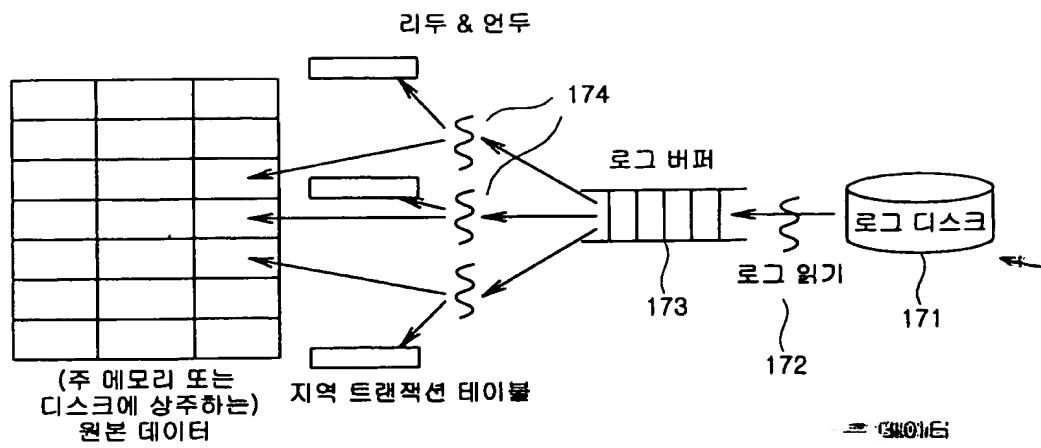
【도 15】



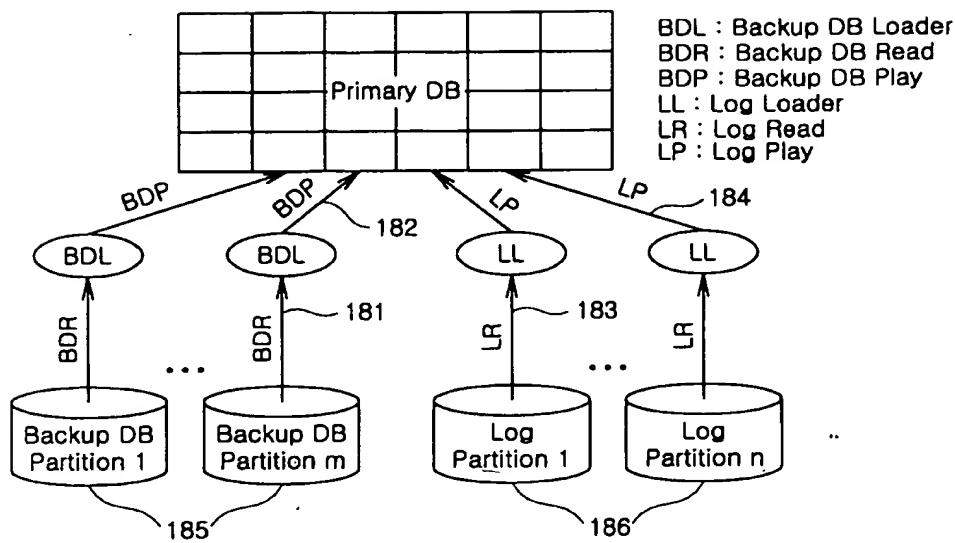
【도 16】



【도 17】



【도 18】



## 【도 19】

